US-PAT-NO: **5872966**

DOCUMENT-IDENTIFIER: US 5872966 A

TITLE: System and method for logging and enabling further manipulation of system state information

---------- KWIC ----------

Abstract Text - ABTX (1):
A client sends a state information message such as an error message or a state information manipulation request to a logging system server. The logging system server prioritizes the messages and requests, and sends them to the appropriate components. A notification engine and a notification manager control notifying the user of the state information message. A **plugin** server and **plugins** control logging the state information message and control other state information manipulation such as generating statistical analysis data and paging a user. The **plugin** server enables additional **plugins** to be added to the system. An external notification manager sends the state information message and possibly other information such as the statistical analysis data to an external system such as technical support. A system fatal error controller locks up the system during a system fatal error to prevent any additional data corruption.

US Patent No. - PN (1):
5872966

Brief Summary Text - BSTX (10):
The present invention overcomes limitations and deficiencies of previous error logging systems by providing an improved system and method for logging and further manipulating system state information. The invented system includes a central processing unit (CPU) that connects with a memory. The memory stores an operating system program, including a logging system which has a logging system server, a notification engine, a notification manager, a **plugin** server, **plugins,** an external notification manager and a system fatal error controller. The logging server prioritizes state information messages and delivers the prioritized messages to the other components. The notification engine and the notification manager present state information messages to the user. The **plugin** server and **plugins** control logging state information in a system log and control other functions such as for example calling a pager, preparing a statistical analysis, or managing errors. **Plugins** are programs, such as device drivers, for controlling target systems in this case to manipulate system state information. More particularly, a log **plugin** which includes a self-initialization routine and adheres to a predetermined protocol for communicating with the **plugin** server gathers and adds system state information to a system log stored in memory. Other **plugins,** each of which also includes a self-initialization routine and adheres to a predetermined

protocol for communicating with the **plugin** server, may be added to the system for performing other functions. The external notification manager forwards state information, and possibly other information such as statistical data, through a communications interface to external systems such as technical support. The system fatal error controller manages system fatal errors, by locking the system and forcing system reboot.

Brief Summary Text - BSTX (11):

The present invention also provides a method for initializing the system including the **plugins,** a method for logging state information in a system log and enabling other manipulation of state information messages, a method for retrieving data from the system log, and a method for processing administrative type requests on the state information messages of the system log.

Brief Summary Text - BSTX (12):

The system and method of the present invention add system state information to a system log to provide a more comprehensive and detailed mechanism for recording system state information. Further, the system and method provide an improved mechanism for handling system errors and for notifying the users, technical support persons and others. Still further, the system and method provide more detailed explanations of the state of the system to users and technical support persons, and add the more detailed explanations to the log for subsequent analysis. Even further, the system and method enable further manipulation of the system state information such as clearing the system log, sizing the system log to desired specifications, and generating charts for better information presentation. Still even further, the system and method use self-initialization routines to provide a mechanism for adding after manufacturing **plugins** which provide additional functions.

Drawing Description Text - DRTX (4):

FIG. 3 is a flowchart illustrating a method for initializing the FIG. 2 **plugins**;

Detailed Description Text - DETX (5):

FIG. 2 is a block diagram of logging system 190, which includes a logging system server 220, a notification engine 225, a notification manager 230, a **plugin** server 235, a log **plugin** 240, other **plugins** 245, an external notification manager 255 and a system fatal error controller 265. Logging system 190 operates in conjunction with preferably a client 210, an output device 130, data storage 160, target systems 250, a communications interface 170 and external systems 260.

Detailed Description Text - DETX (9):

Logging system server 220 forwards state information to notification engine 225, **plugin** server 235, external notification manager 255 and system fatal error controller 265, and forwards state information manipulation requests to **plugin** server 235. Since during system fatal errors the rest of system 100 is considered unreliable, server 220 as a "specialty function" uses bit block

transfer techniques to deliver system fatal error messages such as "Bomb Boxes" via path 223 to output device 130. A "Bomb Box" is a graphical display image indicating a system fatal error.

Detailed Description Text - DETX (11):

Plugin server 235 is a program in communication with server 220, for receiving state information messages and state information manipulation requests from client 210, and outputting them to log **plugin** 240 and other **plugins** 245. **Plugin** server 235 includes an Application Program Interface (API) 237 for communicating with **plugins** 240, 245. **Plugin** server API 237 enables other **plugins** 245 to be added to system 190 by, for example, placing the **plugins into a plugin** system folder (not shown). **Plugins** 240, 245 are programs, which adhere to a predetermined protocol to communicate with API 237, for controlling a target system to perform a particular function. **Plugins** 240, 245 in general each include a power-up self-initialization routine which enables new **plugins** to be added to the system after its initial configuration. A **plugin** 240, 245 may include a device driver.

Detailed Description Text - DETX (12):

More particularly, log **plugin** 240 is a device driver for adding state information to, and retrieving state information from, a log 270 stored in non-volatile data storage 160. Log 270 is preferably a linked list of a user-selected number of state information messages. An example log 270 containing four messages is illustrated in FIG. 8. The earliest messages on the list, for example Message 1 of FIG. 8, are preferably forced off by the latest messages, for example a new Message 5 (not shown), so the list stores only the current messages (in this example the current four messages).

Detailed Description Text - DETX (13):

Other **plugins** 245 enable additional manipulation of system 100 state information, and may provide functions such as error handling, user notification or technical support notification. **Plugins** 245 control other target systems 250, including communications interface 170, output devices 130 such as disks, pagers or printers, etc. For example, if **plugin** 245 receives an error message indicating that a hard disk drive in system 250 is not responding, the **plugin** 245 may for example redirect data to be stored through communications interface 170 to a network server's hard drive (not shown). Other **plugins** 245 may be supplied by original equipment manufacturers or third party vendors.

Detailed Description Text - DETX (16):

FIG. 3 is a flowchart illustrating a method 300 for system 100 to initialize **plugins** 240 and 245 upon start-up. Method 300 begins in step 310 by allocating space in RAM 150 for **plugins** 240, 245. System 100 in step 320 loads **plugins** 240, 245 from data storage 160 into the allocated space in RAM 150. If **plugins** 240, 245 are stored in a **plugin** system folder (not shown), step 320 retrieves the **plugins** from the folder.

Detailed Description Text - DETX (17):

Each **plugin** 240, 245 in step 330 performs its self-initialization routine. For example, log **plugin** 240 includes the following initialization steps: allocating space in RAM 150 for storing the state information log 270, checking the integrity of the allocated RAM, storing the previously-added messages of log 270 into the allocated RAM, scanning for the oldest and newest entries in log 270, and setting up pointers for these entries. Each other **plugin** 245 will perform its own particular initialization steps as provided by its self-initialization routines. Method 300 then ends.

Detailed Description Text - DETX (18):

FIG. 4 is a flowchart illustrating a method 400 for logging and further manipulating state information messages. Method 400 begins in step 410 with a client 210 communicating a state information message to logging system server 220, which prioritizes and sends the state information message to **plugin** server 235, which determines which of the **plugins** 240, 245 should receive the message. Accordingly, **plugin** server 235 in step 420 sends the message to the appropriate ones of **plugins** 240, 245. Alternatively, **plugin** server 235 can send all incoming messages to all **plugins** 240, 245, in which case each **plugin** 240, 245 can then determine whether the message was intended for it. Alternatively, **plugin** server 235 can store messages until pulled by the **plugins** 240, 245.

Detailed Description Text - DETX (19):

Each **plugin** 240, 245 in step 430 gathers the state information provided by client 210, and based on the **plugin's** function gathers any other needed information such as spreadsheet format, equations, etc. In step 440, each **plugin** 240, 245 performs its function. More particularly, log **plugin** 240 adds the state information message to the system log 270. **Plugin** 240 may provide the user with sufficient time to correct an error condition before storing the message to the system log 270, so that easily corrected errors are not added. Further, log **plugin** 240 may add only selected errors to the system log 270, so that trivial errors such as "printer Off-line" are not added. Each other **plugin** 245 performs its intended function, such as paging a user or other person or creating and delivering a statistical analysis to technical support. Method 400 then ends.

Detailed Description Text - DETX (20):

FIG. 5 is a flowchart illustrating a method 500 for retrieving state information messages from log 270. Method 500 begins in step 510 when a client 210 generates a request to read log 270. The READ request may be to read the entire log 270, to read some specified number such as twenty of the messages in log 270, to read log 270 in reverse chronological order, or otherwise. Client 210 sends the request to logging system server 220, which prioritizes the request for scheduled handling and accordingly delivers it to **plugin** server 235.

Detailed Description Text - DETX (21):

Plugin server 235 in step 520 recognizes the READ request and, if multiple READs are requested, instructs log **plugin** 240 to create an iteration routine

for performing multiple READs. Creating an iteration routine includes retrieving the pointers specifying the address of the first and last relevant entries in log 270. **Plugin** server 235 sends the READ request, and possibly the create-iterator instruction, to log **plugin** 240 which in step 530 reads the specified message. **Plugin** 240 conventionally creates an iterator. When **plugin** 240 in step 540 determines, based on the iteration routine, that all iterations of the READ request have completed, method 500 ends. Otherwise, method 500 returns to step 530.

Detailed Description Text - DETX (23):
A client 210 in step 610 generates and sends an administrative type request to logging system server 220, which prioritizes the request and sends it to **plugin** server 235. In step 620, **plugin** server 240 translates the request and sends it to log **plugin** 240. In step 630, **plugin** 240 performs the administrative request. That is, if the request is for a flush, **plugin** 240 adds the state information messages which are stored in queues to log 270. If the request is for a clear, **plugin** 240 deletes the messages in log 270. If the request is for a re-size, **plugin** 245 re-configures log 270 and re-allocates memory for storing log 270. Method 600 then ends.

Claims Text - CLTX (2):
providing a **plugin,** which includes a computer power-up self-initialization routine for configuring the **plugin** and adheres to a predetermined protocol for receiving state information messages and for controlling the target system;

Claims Text - CLTX (4):
using the **plugin** for receiving a client-generated state information message, and for controlling the target system to manipulate the message.

Claims Text - CLTX (6):
said **plugin** comprises a log **plugin** for adding messages to a message log; and

Claims Text - CLTX (14):
5. The method of claim 1 further comprising, before the step of using the **plugin,** the steps of:

Claims Text - CLTX (18):
transferring the messages from the server to the **plugin** in order of priority.

Claims Text - CLTX (20):
a **plugin** server for receiving a client-generated state information message; and

Claims Text - CLTX (21):
a **plugin,** which includes a self-initialization routine and a service routine adhering to a predetermined protocol, coupled to the **plugin** server for receiving the message from the **plugin** server and for controlling the target system to manipulate the message.

Claims Text - CLTX (23):
the **plugin** comprises a log **plugin** for adding the message to a message log; and

Claims Text - CLTX (27):
a logging system server for receiving the message from the client system, and for transferring the message to the **plugin** server and to the engine.

Claims Text - CLTX (30):
a logging system server for receiving the message from the client system, and for transferring the message to the **plugin** server and to the controller.

Claims Text - CLTX (35):
the **plugin** is a log **plugin** for adding messages to a message log; and

Claims Text - CLTX (36):
upon system re-start the controller requests the log **plugin** to add the message to the message log.

Claims Text - CLTX (40):
a logging system server for receiving the message from the client and for transferring the message to the **plugin** server and to the manager.

Claims Text - CLTX (41):
13. The system of claim 6 further comprising a logging system server for receiving a plurality of messages from the client system, for prioritizing the messages, and for transferring the messages to the **plugin** in order of priority.

Claims Text - CLTX (43):
providing a **plugin,** which includes a self-initialization routine adhering to a predetermined protocol, for receiving state information messages and controlling a target system;

Claims Text - CLTX (44):
executing the self-initialization routine for configuring the **plugin**; and

Claims Text - CLTX (45):

using the **plugin** for receiving a client-generated state information message, and for controlling the target system to manipulate the message.

Claims Text - CLTX (47):
a **plugin** server coupled to the client for receiving client-generated state information messages

Claims Text - CLTX (48):
a **plugin,** which includes a self-initialization routine adhering to a predetermined protocol, coupled to the **plugin** server for receiving a client-generated state information message and for controlling the target system to manipulate the message; and

Claims Text - CLTX (49):
a processing unit for executing the self-initialization routine to configure the **plugin**.

Claims Text - CLTX (53):
a **plugin** server for receiving a client-generated state information message; and

Claims Text - CLTX (54):
a **plugin** which uses a predetermined protocol for communicating with the **plugin** server and which controls the target system to perform the particular function on the state information message; and

Claims Text - CLTX (57):
the **plugin** includes a log **plugin** for adding the message to a message log; and

Claims Text - CLTX (61):
a logging system server for receiving the message from the client, and for transferring the message to the **plugin** server and to the notification engine.

Claims Text - CLTX (64):
a logging system server for receiving the message from the client and for transferring the message to the **plugin** server and to the controller.

Claims Text - CLTX (69):
the **plugin** is a log **plugin** for adding messages to a message log; and

Claims Text - CLTX (70):
upon system re-start the controller requests the log **plugin** to add the

message to the message log.

Claims Text - CLTX (75):
a logging system server for receiving the message from the client and for transferring the message to the **plugin** server and to the manager.

Claims Text - CLTX (77):
a **plugin** server for receiving state information messages from clients; and

Claims Text - CLTX (78):
a log **plugin** coupled to the **plugin** server for creating a system state information log and for adding the message information to the log.

Claims Text - CLTX (82):
sending the message based on its priority to **plugins**; and

Claims Text - CLTX (83):
using the **plugins** to manipulate the message.

Claims Text - CLTX (86):
delivering the Request to a log **plugin**;

Claims Text - CLTX (87):
using the log **plugin** to create an iteration routine based on the READ request; and